Controlling Position

Servomechanisms are of this form:

Some General Features of Servos:

- They are feedback circuits
- Natural frequencies are 'zeros' of $1 + G(s)H(s)$
- System is unstable if these zeros are in the right half plane
- 'Negative' feedback becomes positive with 180 degree phase shift
- Putting an integrator into $H(s)$ drives steady error to zero
- But high order systems are more likely to have RHP zeros
- Time delay and high gain lead to RHP zeros

Digital Servos

- Major parts of the system are digital
  - Digital systems are more flexible to design
  - More repeatable (not subject to gain drift)
- Analog parts are important
  - But in many cases can be avoided...
  - But note that digital servos have fixed (or worse stochastic)
    time delays

Position Measurement

- Voltage proportional to position
  - Linear or rotary potentiometer
  - Accuracy limited by that of potentiometer
  - Accuracy limited by voltage source
Measure Rotary Position

Two sinusoidal potentiometers
\[ V_1 = V_0 \cos \theta \]
\[ V_2 = V_0 \sin \theta \]
This can be done magnetically too
Sinusoidal coupling
Requires complex analog detection
Is called a Resolver

These are still analog
Accuracy limited
Subject to drift
Complex calculations

Another Digital Method

Two-Phase Encoder
Two Source-Sensor Sets
Offset by half sector width
This example has 30 degree sectors
And 15 degree resolution

Digital Measurement of Position

Sense light transmission
Typically through a transparent sector
Gives a reading over a range of positions
May need a lot of sensors...
Use of Encoder

This circuit generates:
- An Up/Down signal (CW or CCW)
- A Count Signal (Edge Encountered)

Another way of doing an encoder

- Displace Sensors by 1/2 band
- Add a 'Home' row with an absolute sense

Waveforms

A and B are sensor signals
Rotating one way, Count Edge is when U/D is high
Rotating other way, is when U/D is low

Motors

Simple servomechanisms are made with DC motors
DC Motor Model is simple:
- Resistance in series with a voltage source
- Motor produces torque
- Mechanical system (controlled system) determines speed as influenced by torque
Permanent Magnet DC Motors

Very commonly used:
- 'Back Voltage' proportional to speed
- Torque proportional to current

Servo Strategy:
- Command torque by setting current
- Measure speed

Running open loop:
- There is a 'zero torque' speed
- Torque proportional to difference from that speed

Stepper Motors

Digital Motors
Two "stacks" (phases)
Usually biased by permanent magnets
Move a discrete distance per 'step'
This is an axial view
Cut through both of two sections

Stepper Motor Windings

Two distinct 'phases'
May be driven as distinct windings
Or may be driven as 'bifilar' windings
Bifilar is easier but less efficient

Bipolar Winding

Driven by 'H-Bridges' of transistors
- Can put current through windings in either direction
- But note upper transistor gating is tricky
- Uses all of the winding
Bifilar Winding

- Driven by four transistors to ground
- Note it center of winding is held 'high',
- Transistors are between winding and ground
- NPN bipolar work well
- Transistor gates are easily handled

Motors run in either direction

- Current drive strategy
  - Bipolar winding on left
  - Bifilar winding on right

Dynamics are important

- Stepper can hold a certain torque
- Stepper can carry more torque at low speed
- At high speed, torque must be de-rated
- Motors draw CURRENT!
  - Need to make sure devices can handle it.

Quiz 2

- Walker Room 50-340, Friday, November 1, Class Hour (1-2)
- Two Crib Sheets Allowed
  - (The one for quiz 1 and another one)
- Venue: Lectures 1-16
- Homework Sets 1-5
- Labs 1-3
Finite State Machines:
- State Description
- Transition Tables
- Implementation in hardware
- Implementation in VHDL

VHDL
- Entity Declaration
- Architecture Specification
- Process to wrap around concurrent statements
- Assignments and logical statements
  - if-then, elsif-then, else, endif
  - case-when-end case
- Component instantiation
- How to code FSM's
- Implicit and explicit registers
- Timing implications

DESIGN RULES:
- Use modularity:
  - Small subsystems are simpler to design
  - Subsystem definition is important
- Design for testability:
  - Design subsystems so they will run alone
  - Avoid trap states (check use of Don't Cares)
  - Do your logic design carefully, and first:
    - Avoid problems from "glitches":
      - Gate delays and multiple bit transitions can (and do) cause "glitches".
      - CLK, G, /PR and /CL inputs must NOT have glitches.
      - Carry from counter (e.g. 163) can have glitches.
      - Don't gate the clock.
- Use proper timing:
  - Be sure that combinational output is stable before assertion of clock.
  - Clock period $\geq$ Max (FF delay, Input Changes) + CL delay + Setup.
  - Obey flip-flop timing restrictions: setup, hold times, clock width.
  - Don't derive asynchronous clear from flip-flops to be cleared.
  - All edge-triggered flip-flops must operate on the SAME clock edge.
  - Beware of clock skew.
  - Tree structure to expand clock.
  - Change inputs only (just) after the clock edge.

Functionality (and examples) with which to be familiar:
- Basic AND, OR, NAND, NOR, XOR,...
- Clocked D, J-K flip flops and registers
- Counters ('163, '169 and '393)
- MUXs and Decoders ('151 and '138)
- Shift Registers ('95 and '194)
- Digital comparator and ALU ('85 and '181)

Binary Arithmetic
- Representation of numbers
  - unsigned
  - sign/magnitude
  - Two’s complement
- Addition, subtraction, multiplication

PLD’s:
- PALs and CPLD’s
- Direct implementation of product terms
- Output architectures

Be careful about asynchronous events:
- Synchronize all external inputs.
  - Asynchronous event should change ONLY one flip-flop.
- Avoid tri-state bus contention.
- Don't overload outputs (observe fan-out).
- Use memory properly:
  - Avoid High-Z address to SRAM when CE is true.
  - Avoid address changes when write pulse is true.
  - Make sure your write pulse is "clean".
- Wire properly:
  - Keep wires short.
  - Wire all inputs (even unused ones).
  - Use bypass (decoupling) capacitors.
  - Use multiple grounds between kits
  - Alternate ground with signals in flat cables. Or use twisted pairs between kits.
  - Don't overload your power supply!
- Use debugging strategy:
  - Debug modules systematically
  - Is every pin wired? Why not?
  - Use a scope! Check valid logic levels and power supply.
  - Use your logic analyzer for checking sequencing